

ADQL - PEG grammar

Progress status

IVOA Interoperability Meeting
16 Nov. 2024 - Malta

Grégory Mantelet





Remember!

Poster P919

ADASS XXXIII in Tucson

Novembre 2023

G. Mantelet, M. Demleitner, J. Juaristi Campillo



PEG-ify ADQL

SELECT CAST(grammar AS peg) FROM adql

Grégory Mantelet (CDS)
gregory.mantelet@astro.unistra.fr
Markus Demleitner (GAVO),
Jon Juaristi Campillo (GAVO)



What is ADQL?

ADQL stands for Astronomical Data Query Language.

This language is defined by the IVOA. It is a fork of SQL-92 in which astronomical functions and operators have been added.



IVOA's ADQL Recommendation
<https://www.ivoa.net/documents/ADQL/>

```
-- Display basic data about objects contained in a given cone:
SELECT DISTANCE(POINT('ICRS', RA, DEC),
  POINT('ICRS', 10, 5)) AS 'Distance from cone center',
  main_id AS 'Main identifier',
  objtype AS 'Object type',
  ra, dec,
  pmra, pmdec
FROM basic
WHERE CONTAINS(POINT('ICRS', RA, DEC),
  CIRCLE('ICRS', 10, 5, 1)) = 1
ORDER BY 1 ASC;
```

Figure 1: ADQL query running a cone search query on Simbad's TAP service at <http://simbad.cds.unistra.fr/simbad/sim-top>



Interesting. How is this language described?

ADQL, as many languages, is described by a grammar. Since Version 2.0, the IVOA provides the ADQL grammar using the *BNF notation*.

However, this notation has some limitations. We'd like to try using a PEG one instead.

Can you help?



BNF notation

- Backus Naur Form or Backus Normal Form
- Created by John Backus and Peter Naur in 1960
- Context Free Grammar (CFG)
- Multiple variants of BNF: EBNF, ABNF, ...



```
<select_query> ::=
  SELECT
  [ <set_quantifier> ]
  [ <set_limit> ]
  <select_list>
  <from_clause>
  [ <where_clause> ]
  [ <group_by_clause> ]
  [ <having_clause> ]
```

Figure 2: Excerpt of the BNF for ADQL



Why changing?

- ADQL's BNF is not a machine readable variant of BNF
 - no parser is able to read/validate it
- Unclear token separation (e.g. is a space needed?)
 - e.g. `SELECT 23a` = number with typo, or `SELECT 23` as a
- Ability to deal with ambiguities of natural languages
 - makes the grammar unnecessarily more complicated for a machine-oriented language



Of course.

PEG stands for Parsing Expression Grammar. It is introduced by Bryan Ford in 2004.



"Parsing Expression Grammars: A Recognition-Based Syntactic Foundation"
Bryan Ford, 2004, doi:10.1145/964001.964011
<https://bford.info/pub/lang/peg.pdf>

As opposed to BNF, it is a notation entirely dedicated to machine-oriented languages. It is not designed to be able to deal with ambiguous expressions of natural languages like CFG and BNF do.



Draft ADQL-2.1 PEG

<https://github.com/ivoa/lyonetia/blob/master/src/peg/adql2.1.peg>



Features

- ✓ **Prioritized choice:** no more choice between two possible rules: the 1st matching one in grammar order always applies.
- ✓ **Combined tokenisation and parsing** in one step
- ✓ **Regular expression style** with *, +, ?, ...

```
select_query <-
  SELECT
  [ ... set_quantifier? ]
  [ ... set_limit? ]
  - select_list
  table_expression <-
  from_clause
  [ ... where_clause? ]
  [ ... group_by_clause? ]
  [ ... order_by_clause? ]
  [ ... offset_clause? ]
```

Figure 3: Excerpt of the draft PEG for ADQL



Parsers Generators

Bryan Ford's Packrat package (<https://bford.info/packrat/>) lists a lot of parsers in multiple programming languages. Some are outdated though.

Here are the parsers we started to look at:

- ✓ Mouse (Java)
- ✓ Arpeggio (Python)
- ✓ peg/leg (C)
- ✓ PEG.js (Javascript)
- ✓ Canopy (Java, Javascript, Python and Ruby)



A problem...

The syntax accepted by PEG parsers often differs from one implementation to another.

Examples:

- **Rule separator:** <- in Ford PEG, <- ... ; or = in Arpeggio, = in Mouse
- **Comments:** # in Arpeggio and Ford PEG, // or /*...*/ in Mouse,
- **Non-matching syntax:** `! [^a-zA-Z]` in Arpeggio, `! [a-zA-Z]` in Ford PEG, `^[a-zA-Z]` in Mouse, `[^a-zA-Z]` in peg/leg
- **Identifier syntax:** `camelCase` in Mouse, `snake_case` in Arpeggio

A solution: write the ADQL PEG grammar following the Ford PEG notation and then write converters to target languages.

Next steps

- Choose a PEG syntax
- Update the PEG to ADQL 2.1-REC (and squash remaining bugs)
- Write a validator based on PEG
- Test all validation queries collected in GitHub ivoa/lyonetia
- Write converters from this grammar to some target parsers

Next developments in

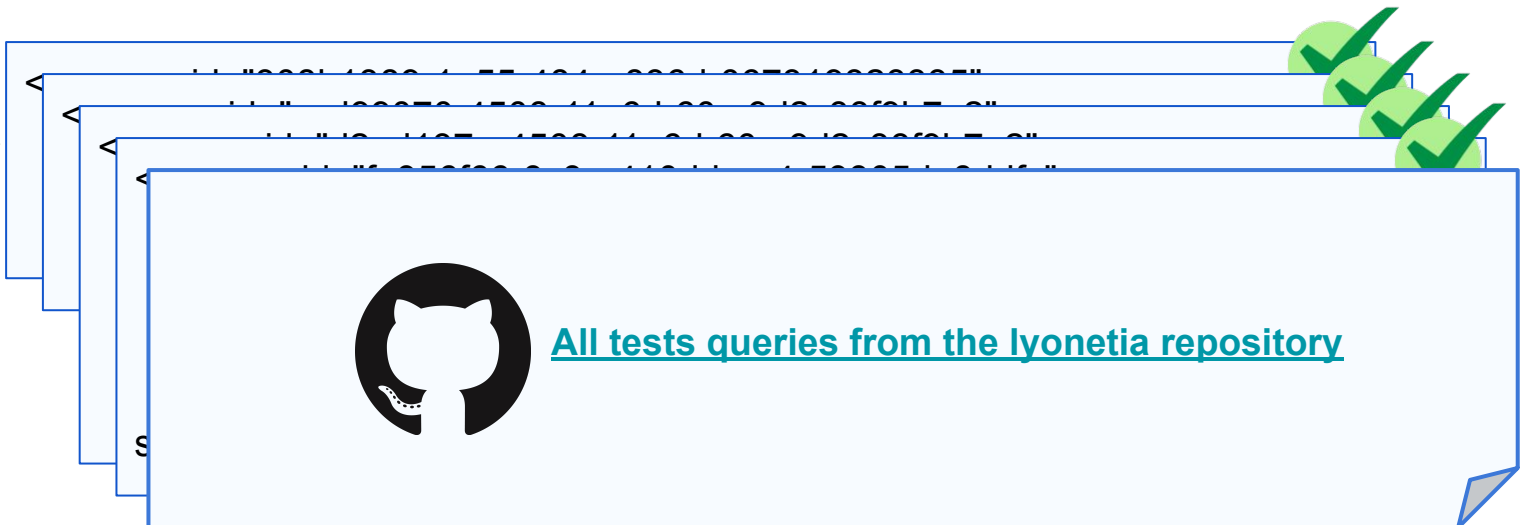
- GitHub ivoa/lyonetia
- PEG grammar + validator
- GitHub ivoa-std/adql
- standard + final grammar



□ The goal is to validate the ADQL grammar

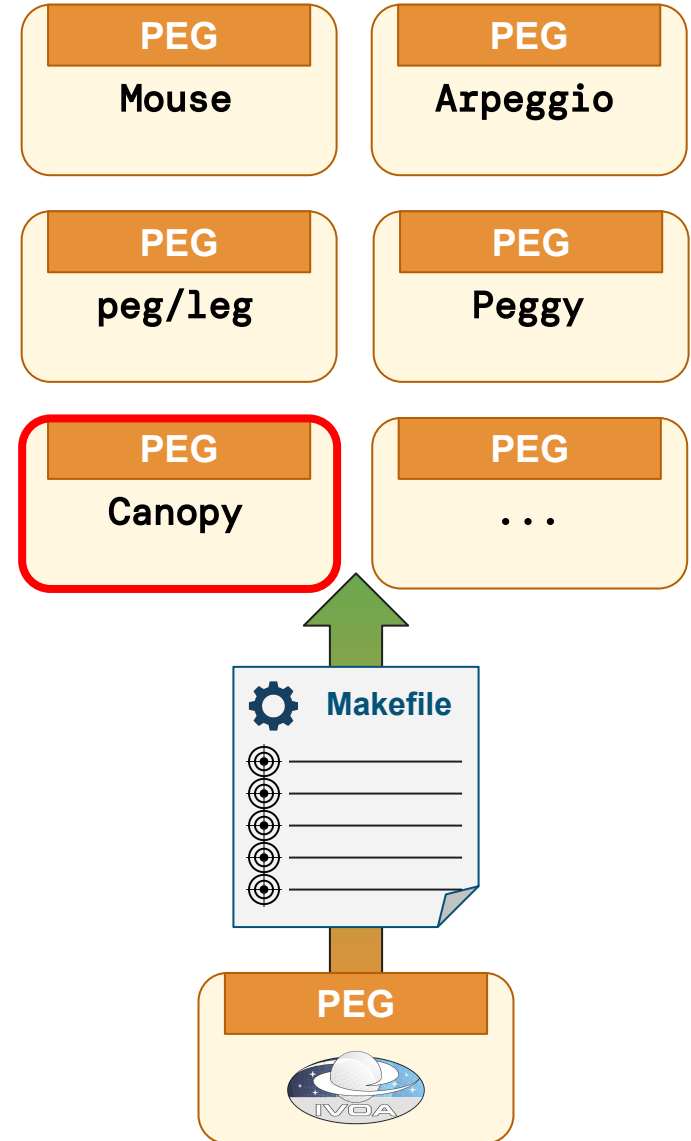
PEG


```
QuerySpecification <- WithClause? _  
                    QueryExpression _ EOF  
  
QueryExpression <- SelectQuery  
                  ( _ set_operator  
                    _ set_query_expression )*  
  
[...]
```



□ We need conversions for existing PEG flavors

	Java	Python	C	JS	Ruby
<u>Mouse</u>					
<u>Arpeggio</u>					
<u>peg/leg</u>					
<u>Peggy</u>					
<u>Canopy</u>					





**Do you need to deal with
ADQL queries in other
languages or with other
tools?**

□ Build the PEG grammar snippet by snippet

Aa Typography

! Fix issues

📄 Put all together

✓ Run all tests

1. Take the [draft PEG grammar](#)
2. Fix typography
(e.g. CamelCase,
recipes alignment, ...)

- left recursion in:
 - column names
 - table names
 - schema names
 - math expressions
- identifiers != reserved
-

Put all snippets into the final ADQL grammar.

Validate [all test queries of lyonetia](#) with this PEG grammar generated for Canopy+Java.

□ Next: getting closer to ADQL-2.2

1. Review and fix PEG grammar
2. Generate Java parser with Canopy
3. Validate tests queries
4. Publish grammar + Makefile on GitHub
5. Support other parser generators

thank you

